Active Learning for Directed Exploration of Complex Systems

Michael C. Burl MICHAEL.C.BURL@JPL.NASA.GOV
Jet Propulsion Laboratory, California Institute of Technology, 4800 Oak Grove Drive, Pasadena, CA 91109

Esther Wang ESWANG@CALTECH.EDU
California Institute of Technology, 1200 E. California Blvd, Pasadena, CA 91125

Abstract

Physics-based simulation codes are widely used in science and engineering to model complex systems that would be infeasible to study otherwise. Such codes provide the highest-fidelity representation of system behavior, but are often so slow to run that insight into the system is limited. For example, conducting an exhaustive sweep over a d-dimensional input parameter space with k-steps along each dimension requires k^d simulation trials (translating into k^d CPU-days for one of our current simulations). An alternative is *directed exploration* in which the next simulation trials are cleverly chosen at each step. Given the results of previous trials, supervised learning techniques (SVM, KDE, GP) are applied to build up simplified predictive models of system behavior. These models are then used within an active learning framework to identify the most valuable trials to run next. Several active learning strategies are examined including a recently-proposed information-theoretic approach. Performance is evaluated on a set of thirteen synthetic oracles, which serve as surrogates for the more expensive simulations and enable the experiments to be replicated by other researchers.

1. Introduction

Many complex processes cannot be studied directly for a variety of reasons including: cost, safety, non-repeatability (e.g., destructive testing), lack of control over relevant variables (e.g., conditions affecting global climate), inability to replicate appropriate conditions in a laboratory environment, and so on. Physics-based numerical simulations offer the only practical means to model and study many of these systems. The internal logic of a simulation code frequently takes the form of coupled partial differential/difference equations that describe the interactions between a large set of elemental nodes or particles. For example, the planetary physics simulations of Durda et al. (2004) use a Smooth Particle Hydrodynamics (SPH) code to model the propagation of energy and fragmentation during the initial collision between two asteroids followed by an efficient N-body gravitational code to model the subsequent interactions between the resulting fragments.

For our purposes, we are not so interested in the detailed internals of the simulation code; we simply treat the simulator as a black box that takes input parameters and produces raw output. If necessary, a "grading script" can be introduced to evaluate and transform the raw output of the simulator into a binary-valued decision indicating whether the simulation trial produced a desired outcome ($+1$) or not (-1). Different grading scripts can address various scientific questions within the scope of a single simulator; however, to simplify the terminology, we will take "simulator" to mean "simulator-grading script combination".

A toy problem considered in the experiments is the classic inverted pendulum on a cart studied in control theory (Kwakernaak & Sivan, 1972). The goal is to balance the stick (inverted pendulum) in a vertical position by moving the cart back and forth. The input to this "simulator" consists of four state feedback gain coefficients, and the output is a $+1$ if the resulting controller succeeds in balancing the stick and a -1 if it fails. Although in this example a learner could plausibly receive a real-valued signal indicating how close the control system was to succeeding (or failing), we restrict our attention to the situation in which *the*

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009.

learner is only informed of a binary-valued outcome, as this assumption better fits our actual problem. Using simulations and *directed exploration*, we attempt to efficiently learn the set of all controllers that stabilize the system (produce a +1 output).

1.1. Related Work

Active learning has received considerable attention in the machine learning community dating back to the work of MacKay (1992) and Cohn et al. (1994). These techniques have been applied with some success to the problem of efficiently learning a predictive model of a continuous-valued output, i.e., in the regression setting. For example, Gramacy et al. (2004) used non-stationary Gaussian process trees to explore a computational fluid dynamics simulation of a NASA reentry vehicle. Pfingsten (2006) used a Bayesian active learning technique to assist in analyzing micro-mechanical sensors. A research area known as Design and Analysis of Computer Experiments (DACE) (Sacks et al., 1989) uses statistical methods, including kriging, to construct surrogates to deterministic computer models. However, outside of our own pilot work (Burl et al., 2006), there has been little effort directed toward using active learning to efficiently explore simulations that provide binary-valued outputs.

The directed exploration problem arises not only for simulators, but also for physical systems that are capable of autonomously performing experiments. Knuth et al. (2007) considered a problem in which a robot arm with a light sensor sequentially makes measurements within its workspace to efficiently determine the size and location of a target region. Nested sampling and Bayesian adaptive estimation were used, but the technique has not been demonstrated outside of a fairly limited hypothesis space consisting of circular disks. Their approach is closely related to the Bayesian Experimental Design philosophy used in (Veeramachaneni et al., 2006; Olivetti, 2008). Guestrin et al. (2005) consider the problem of optimal sensor placement using Gaussian Processes and a mutual information criterion. They conclude that the sensors should be spread evenly throughout the domain since the posterior variance in their predictions does not depend on the observed sensor values. However, in the classification setting that we consider, the observed "sensor values", i.e., the outcomes of simulator trials, are critical in deciding where to sample next.

2. Approach

The simulator takes as input a vector of parameters, $\boldsymbol{\theta}$, defined over a continuous-valued input domain, $\boldsymbol{\Theta}$, and, with the help of a grading script, outputs a binary-valued indicator of whether the simulation trial produced a desired outcome (+1) or not (−1). We denote the end-to-end function from input to output as $q(\boldsymbol{\theta})$. It is well known that some dynamical systems are chaotic; for such systems small variations in $\boldsymbol{\theta}$ could produce very different values from q . We do not consider such systems, nor do we consider noisy systems where the value returned for a given input $\boldsymbol{\theta}$ varies from one run to the next. Instead, we assume a deterministic simulator that has some underlying spatial coherence (spatial correlation, spatial structure, continuity) in the $q(\cdot)$ function.

2.1. Approximation from Samples

The Nyquist Sampling Theorem, a fundamental result from signal processing, states that a *band-limited* signal can be exactly represented by its samples provided that the sampling rate is high enough. Because of our spatial coherence assumption, an oracle will consist mainly of low spatial frequency regions (except, of course, for step edges at the transition between areas of +1 and −1); hence, there is some hope that the function can be well-approximated through a finite set of samples. Results from computational learning theory, such as the Support Vector Machine (SVM) of Vapnik (1995), also suggest that one can learn any well-behaved function from a set of training data. If all of the grid points and corresponding labels from the example above are supplied to an SVM learning algorithm as training data, a relatively small subset of the data, i.e., the support vectors, will be sufficient to define a function that agrees almost everywhere with the true function $q(\cdot)$.

For the simulation problem, we want to construct an approximation, $\hat{q}(\boldsymbol{\theta})$, to $q(\boldsymbol{\theta})$ such that $\hat{q}(\cdot)$ agrees with $q(\cdot)$ over most of the input domain. Further, the function $\hat{q}(\cdot)$ will be defined from training examples: input-output pairs that are in essence samples in the signal processing sense[1] from the true function $q(\cdot)$. Although one could select these samples through (i) a grid strategy in which trial points are spread evenly across the input domain or (ii) a random strategy in which trial points are chosen at random within the domain, we believe that a more informative set of points can be *actively* selected by taking into account the current state of knowledge.

Thus, our basic approach consists of predictive modeling plus active learning. The physics-based simulation is used as an oracle to sequentially gener-

[1] There is no constraint, however, that the samples be regularly spaced.

ate labeled training data in the form of input-output pairs. At each step or round, the algorithm uses a supervised learning algorithm to form a simplified predictive model based on currently available training data. However, because the simulations are so computationally-demanding, it is critical to get the most out of each simulation trial. An active learning algorithm then chooses an unlabeled point that would be particularly valuable if it were labeled (by a run through the simulator).

2.2. Predictive Models

The predictive model, $\hat{q}(\cdot)$, is supposed to approximate the input-output behavior of the simulator. As more training data is acquired through ongoing runs of the simulator, our estimate for $\hat{q}$ will change (hopefully improve!); a superscript (r) will be added to $\hat{q}$ to designate the estimated surrogate function after the r-th example from the simulator has been observed. Also, since we recover $\hat{q}(\cdot)$ through a learning procedure that typically depends on additional hyperparameters, e.g., kernel type, kernel bandwidth, we use $\hat{q}^{(r)}_{\boldsymbol{\lambda}}(\cdot)$ to designate the estimated function after the r-th trial using hyperparameters $\boldsymbol{\lambda}$.

Although the surrogate function will ultimately produce binary-valued labels $(+1)$ or (-1), we prefer to think of $\hat{q}(\cdot)$ as producing a continuous-valued output that can be thresholded at different values to produce a receiver operating characteristic (ROC) curve. We consider three different types of predictive models for the surrogate function: support vector machines (SVM), kernel density estimation (KDE), and Gaussian processes (GP).

2.2.1. Support Vector Machines (SVM)

Since their introduction by Vapnik, SVMs have been extensively studied. Learning an SVM from training data involves solving a quadratic programming problem. We use the `libsvm` implementation (Chang & Lin, 2001) in our work. Hyperparameters for the SVM learning problem include the type of kernel (RBF, polynomial, etc.), any kernel specific parameters (RBF bandwidth), and the regularization constant (C). The learned model consists of some number n_s of support vectors $\mathbf{s}_i$, weights β_i, and a scalar offset b. The decision statistic is defined as:

$$\hat{q}(\boldsymbol{\theta}) \quad = \quad -b + \sum_{i=1}^{n_s} \beta_i K(\boldsymbol{\theta}, \mathbf{s}_i) \tag{1}$$

where $\boldsymbol{\theta}$ is a point in input space and $K(\cdot,\cdot)$ is the kernel function. Note that we have omitted the usual $\mathrm{sgn}(\cdot)$ function around the right hand side to preserve the continuous-valued decision statistic. We could use a logistic function as in (Platt, 1999) to transform this into a pseudo-probability.

2.2.2. Kernel Density Estimation (KDE)

With kernel density estimation (KDE), also known as the Parzen Window Technique, we attempt to directly estimate the posterior probability that a point belongs to the positive or negative class by modeling the class-conditional probability densities. KDE estimates the posterior probability for the label t of a point $\mathbf{z}$ as follows:

$$p(t=+1|\mathbf{z},\mathbf{X},\mathbf{y}) \quad = \quad \frac{S_+(\mathbf{z})}{S_+(\mathbf{z}) + S_-(\mathbf{z})} \tag{2}$$

where $S_+(\mathbf{z}) = \alpha_+ + \sum_{\mathbf{X}_+} K(\mathbf{z}, \mathbf{x}_i)$ and $S_-(\mathbf{z}) = \alpha_- + \sum_{\mathbf{X}_-} K(\mathbf{z}, \mathbf{x}_j)$. $\mathbf{X}_+$ is the set of instances in $\mathbf{X}$ that have $+1$ labels and $\mathbf{X}_-$ is the set of instances in $\mathbf{X}$ that have -1 labels. The scalar values α_+ and α_- are used in Bayesian fashion to define a prior probability that must then be supported (or overturned) by the actual data. Active learning requires frequent update of the classifier given a newly selected training example. It is clear from Equation 2 that KDE allows a simple recursive update.

2.2.3. Gaussian Processes (GP)

Let r be a (hidden) random variable that represents a continuous-valued version of the label t of $\mathbf{z}$. Under a GP model, the posterior probability density over r given $\mathbf{z}$ and the training set is:

$$p(r|\mathbf{z},\mathbf{X},\mathbf{y}) \quad = \quad \mathcal{N}\left(r;\ \mu_{r|\mathbf{z},\mathbf{X},\mathbf{y}},\ \sigma^2_{r|\mathbf{z},\mathbf{X},\mathbf{y}}\right) \tag{3}$$

where $\mathcal{N}$ represents the normal distribution. The conditional mean and covariance are as follows:

$$\mu_{r|\mathbf{z},\mathbf{X},\mathbf{y}} \quad = \quad K(\mathbf{z},\mathbf{X}) \cdot K(\mathbf{X},\mathbf{X})^{-1} \cdot \mathbf{y} \tag{4}$$

$$\sigma^2_{r|\mathbf{z},\mathbf{X},\mathbf{y}} = K(\mathbf{z},\mathbf{z}) - K(\mathbf{z},\mathbf{X})K(\mathbf{X},\mathbf{X})^{-1}K(\mathbf{X},\mathbf{z}) \tag{5}$$

Integrating the posterior probability distribution over r from 0 to ∞ provides a probability for the event $\{t=+1\}$.

$$p(t=+1|\mathbf{z},\mathbf{X},\mathbf{y}) \quad = \quad \int_0^\infty p(r|\mathbf{z},\mathbf{X},\mathbf{y})\, dr \tag{6}$$

$$= 1 - \Phi(0;\ \mu_{r|\mathbf{z},\mathbf{X},\mathbf{y}}, \sigma^2_{r|\mathbf{z},\mathbf{X},\mathbf{y}}) \tag{7}$$

where Φ is the cumulative distribution function of the Gaussian distribution. As with KDE, the GP classifier allows for a relatively straightforward recursive update (in particular, the inverse covariance can be efficiently updated using the Block Matrix Inversion Lemma). The overall cost is roughly $O(PM + P^2)$, where P is the number of training examples and M is the number of different values of $\mathbf{z}$.

2.3. Active Learning Strategies

2.3.1. Passive Learning (PAS)

In passive learning, the next point is randomly chosen from the remaining set of unlabeled points. In contrast, in active learning we choose the next point according to some value criterion.

2.3.2. Most Confused Point (MCP)

A common active learning strategy (Tong & Koller, 2001) chooses the unlabeled point whose label is most uncertain given the current state of knowledge. Let $\hat{q}^{(r)}(\boldsymbol{\theta})$ represent the probability that $\boldsymbol{\theta}$ should be assigned to class $+1$ given information up to round r. Then an MCP strategy would choose:

$$\boldsymbol{\theta}_{\text{MCP}} \quad = \quad \arg\min |\hat{q}^{(r)}(\boldsymbol{\theta}) - 0.5| \tag{8}$$

(For SVM, this expression assumes the raw hyperplane distance has been converted into a pseudo-probability.)

Alternatively, we could convert the probabilities into a single point entropy

$$\mathcal{H}(p) = -p \log_2 p - (1-p) \log_2 (1-p) \tag{9}$$

and then select the point with maximum entropy.

2.3.3. Most Informative Point (MIP)

Even if the label of a particular point is highly uncertain, knowledge of its true label may have little leverage over other unlabeled points. Consider a point A that falls precisely between a well-understood region of positive examples and a similar region of negative examples. Knowing the label of A may not be very valuable because A does not have much influence over the other unlabeled points. Now consider a point B that is in an unexplored cluster of points far away from any currently labeled points. Knowing the label of B may be extremely valuable because the other points in the cluster would presumably belong to the same class. Thus, knowing this label simultaneously removes uncertainty about the labels of many other points.

The basic MIP idea, as developed in (Holub et al., 2008), is as follows. Suppose we have a set of training instances $\mathbf{X}$ with known labels $\mathbf{y} \in \{-1, +1\}$. From this training set, we can learn a classifier that makes a probabilistic prediction about the label t of a test point $\mathbf{z}$. The uncertainty about t given the training instances and training labels can be measured with the binary entropy function $\mathcal{H}(\cdot)$ from Equation 9:

$$H(t|\mathbf{z}, \mathbf{X}, \mathbf{y}) \quad = \quad \mathcal{H}(p) \tag{10}$$

where p is the classifier's estimate of the probability that the label of $\mathbf{z}$ is $+1$.

Now, suppose that we have the ability to "look ahead" and add another point $\mathbf{u}$ to the set of training instances (Lindenbaum et al., 2004). We do not yet know the label of this new point, only the probability that $\mathbf{u}$ will have label $v = +1$. We can compute the uncertainty about the label t of $\mathbf{z}$ given the original training instances ($\mathbf{X}$), the training labels ($\mathbf{y}$), and the new training instance ($\mathbf{u}$) along with the hypothesized label $v = +1$. Using the hypothesized label $v = -1$, the uncertainty under the same conditions can be computed. Then, the *expected* uncertainty is given by:

$$\begin{aligned} \overline{H}(t|\mathbf{u}) \quad &= \quad \mathbf{E}_v \left\{ H(t|\mathbf{z}, \mathbf{X}, \mathbf{y}, \mathbf{u}, v) \right\} && (11) \\ &= H(t|\mathbf{z}, \mathbf{X}, \mathbf{y}, \mathbf{u}, v = +1) \cdot p(v = +1|\mathbf{u}, \mathbf{X}, \mathbf{y}) + \\ &\quad H(t|\mathbf{z}, \mathbf{X}, \mathbf{y}, \mathbf{u}, v = -1) \cdot p(v = -1|\mathbf{u}, \mathbf{X}, \mathbf{y}) && (12) \end{aligned}$$

The expected information gain, $\overline{\Delta I}$, about the label t is the expected amount of uncertainty that would be removed by incorporating the new instance $\mathbf{u}$ into the training set:

$$\overline{\Delta I}(t;\, \mathbf{u}) \quad = \quad H(t|\mathbf{z}, \mathbf{X}, \mathbf{y}) - \overline{H}(t|\mathbf{z}, \mathbf{X}, \mathbf{y}, \mathbf{u}) \tag{13}$$

This represents the expected amount of information (in bits) that we will gain about t by incorporating $\mathbf{u}$ into the training set. With multiple unlabeled instances (different points t), we can sum the expected information gain to determine the overall value of incorporating $\mathbf{u}$ into the training set.

Since both KDE and GP directly provide probability estimates for points, we consider using these methods with MIP. Two approaches could be tried to make SVM compatible with MIP. The logistic regression approach of Platt could be used to convert the SVM hyperplane distance into a pseudo-probability. Alternatively, the committee of classifiers approach introduced in (Holub et al., 2008) could be used to construct probabilities from a hard-classifier. However, this approach requires *two* levels of lookahead, making it computationally impractical. We have yet to evaluate either of these approaches for SVM.

2.3.4. Randomization Meta-Strategy (META)

We also considered a meta-strategy in which the base active learning algorithm does not make a definite choice about which point to deliver to the oracle; instead, the base algorithm simply provides a ranking of the points or a weighted preference over the points. The preference is converted to a probability distribution over the unlabeled points. The actual point delivered to the oracle is then picked randomly according to this probability distribution. The META strategy was only evaluated with SVM. Baram and Luz (Baram et al., 2004) discuss other meta-learning strategies.

3. Experimental Results

3.1. Oracles

There is a significant cost in time and computer resources in acquiring simulation data. In many cases, the resulting datasets are proprietary making it difficult to use such data for algorithm development or head-to-head comparisons. Hence, our experiments were conducted using a set of surrogate oracles[2]. All oracles are defined over the domain $\mathbf{\Theta} = [-1, +1]^d$, where d is the dimensionality (number of input parameters). Figure 1 (at the end of the paper) illustrates the datasets.

3.2. Evaluation Methodology

Nine methods were evaluated on the thirteen surrogate oracles shown in Figure 1: SVM-PAS, SVM-MCP, SVM-META, KDE-PAS, KDE-MCP, KDE-MIP, GP-PAS, GP-MCP, GP-MIP. Each method was run seven times on each of the oracles for 400 rounds of active learning. To reduce the run time of the experiments, the active learning algorithms considered only a subset of 100 points from the unlabeled pool at each round.

All three of the supervised learning algorithms used the same RBF kernel ($K(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\gamma||\mathbf{x}_1 - \mathbf{x}_2||^2)$) with parameter $\gamma = 50$. For SVM the regularization parameter was set to $C = 10$. For KDE, the values of α_+ and α_- were set to 0.1. For GP, the value of the noise standard deviation σ_n was set to 0.1.

For each of the methods, a classifier was trained using the sequence of points selected by active learning during round 1 through round r. The resulting classifier, $\hat{q}_\lambda^{(r)}$, was applied over a fine discretization of the full domain of the oracle. At each discretized point, a real-valued number is produced by the classifier. For KDE and GP, this number is an estimate of the posterior probability that the point is in the positive class, while for SVM, the number is the signed hyperplane distance as in Equation 1.

ROC curves are calculated for each method and oracle, as a function of the number of rounds, r, of active learning. Figure 2 shows ROC curves for GP-MCP on Oracle 7. Note that each curve has been resampled using a fixed set of points on the P_{fa} axis from 0 to 1 spaced in steps of 0.0005.

Traditionally, active learning algorithms are evaluated with learning curves that show the classifier accuracy as a function of round. However, our oracles and the simulator problem in general have a significant imbalance between the positive and negative classes, with the negatives greatly outweighing the positives. A classifier that simply says -1 regardless of the input point will already achieve a high degree of agreement with $q(\cdot)$. Instead of using classification accuracy, we use the ROC curves to identify the probability of detection P_d at a fixed false alarm rate and plot P_d versus rounds. Figure 3 shows the modified learning curves for each of the nine methods on Oracle 9.

[2]Matlab code for reproducing these oracles is available by request from the author.

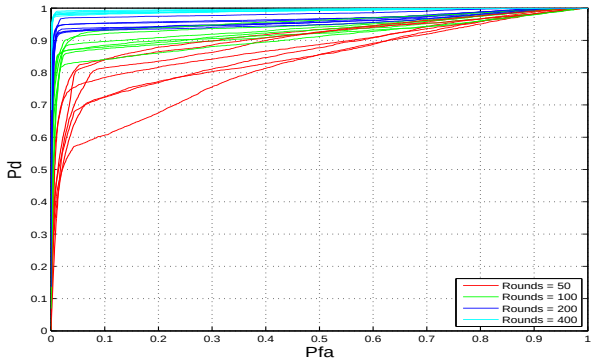

Figure 2. ROC curves on Oracle 7 for GP-MCP, color-coded by number of rounds (r). For each value of r, seven trials were conducted to assess variability.

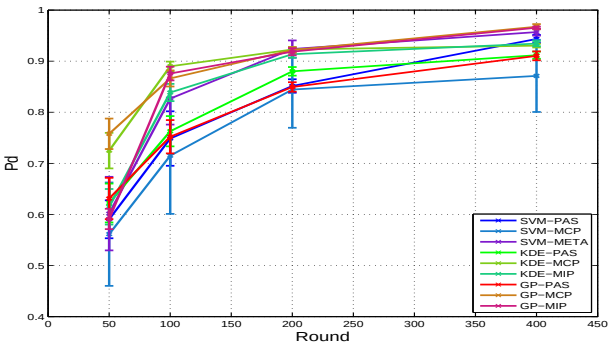

Figure 3. Learning curves for each of the nine methods on Oracle 9. Each curve shows the probability of detection (P_d) at a fixed false alarm probability ($P_{fa} = 0.05$) as a function of the number of rounds of active learning. Except for SVM-MCP, the active methods clearly outperform passive. KDE-MCP reaches 90% P_d at round 100, while KDE-PAS doesn't reach this level until after round 300, representing speedup by a factor of three.

3.3. Discussion

For each of the supervised learning techniques, the results clearly show that active learning outperforms passive learning. For example, in Figure 4, row 1 shows KDE using MIP (information gain) for active learning

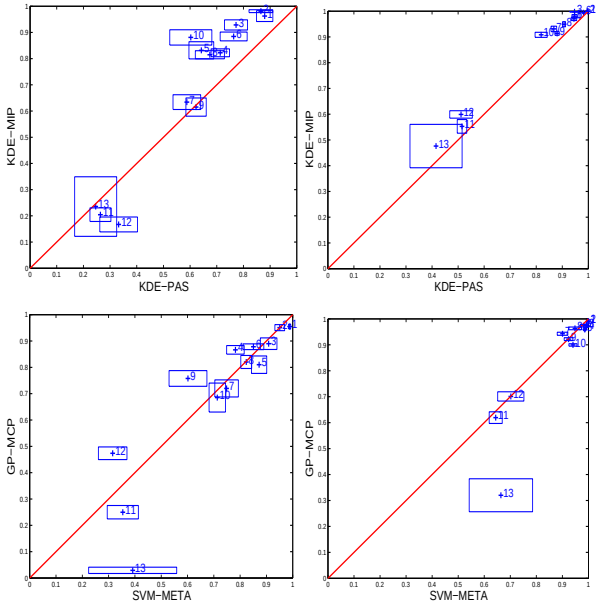

Figure 4. Head-to-head comparison between two methods on all oracles. Each box corresponds to one oracle; box size indicates standard error. Points above the red line favor the y-axis method, below the red line favors the x-axis method. (a) KDE-MIP vs. KDE-PAS ($r = 50$). (b) KDE-MIP vs. KDE-PAS ($r = 200$). (c) GP-MCP vs. SVM-META ($r = 50$). (d) GP-MCP vs. SVM-META ($r = 200$). KDE-MIP dominates KDE-PAS. GP-MCP and SVM-META perform similarly.

is almost always better than KDE using passive learning on all thirteen oracles at rounds 50 and 200.

Looking at additional pairs of box plots (not shown) or at Tables 1-3, we can also conclude that the performance of MCP active learning is approximately the same across the three different supervised learning techniques. Somewhat unexpectedly, MCP appears to be slightly better than the more sophisticated MIP. We hypothesize that MIP may be more sensitive to the accuracy of the estimated probabilities. Another limitation with MIP is that it scales quadratically with the size of the unlabeled pool. For computational reasons, it was necessary to consider a relatively small random subset of the full unlabeled pool at each round, which may have also degraded the performance. A possibility for improving the scalability is to use MCP as a prefilter to prioritize the points that should be evaluated with the more expensive information gain criterion.

SVM-META was typically better than SVM-MCP. In Figure 3, SVM-MCP performs worse than passive learning. We believe that SVM-MCP finds one of the "islands" in Oracle 9 and focuses on refining its boundary rather than adequately exploring the rest of the input domain, i.e., it falls into a sampling selection bias (Zadrozny, 2004) trap.

4. Conclusion

Directed exploration can be used to efficiently understand the behavior of complex systems modeled by numerical simulations. The simulator is used as an oracle to sequentially generate training data (input-output pairs); a simplified predictive model of the system is learned from the currently available training data and this model is used in an active learning framework to choose the best simulation trials to run next. Performance evaluation using active learning on a challenging set of oracles showed a significant improvement over passive learning, in some cases showing a speedup by a factor of three in the number of rounds required to reach 90% probability of detection at a false alarm rate of 0.05.

Acknowledgment

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. ©2009 California Institute of Technology. Government sponsorship acknowledged. We thank collaborators B. Enke, W.J. Merline, P. Perona, and A. Holub, as well as the anonymous reviewers.

References

Baram, Y., El-Yaniv, R., & Luz, K. (2004). Online choice of active learning algorithms. *J. Machine Learning Research*, *5*, 255–291.

Burl, M. C., DeCoste, D., Enke, B. L., Mazzoni, D., Merline, W. J., & Scharenbroich, L. (2006). Automated knowledge discovery from simulators. *SIAM Int. Conf. on Data Mining* (pp. 82–93).

Chang, C.-C., & Lin, C.-J. (2001). LIBSVM: A library for support vector machines. `http://www.csie.ntu.edu.tw/cjlin/libsvm`.

Cohn, D., Atlas, L., & Ladner, R. (1994). Improving generalization with active learning. *Machine Learning*, *15*, 201–221.

Durda, D. D., Bottke, W. F., Enke, B. L., Merline, W. J., Asphaug, E., Richardson, D. C., & Leinhardt, Z. M. (2004). The formation of asteroid satellites in large impacts: results from numerical simulations. *Icarus*, *170*, 243–257.

Gramacy, R. B., Lee, H. K., & Macready, W. G.

(2004). Parameter space exploration with gaussian process trees. *Int. Conf. on Machine Learning* (pp. 353–360).

Guestrin, C., Krause, A., & Singh, A. P. (2005). Near-optimal sensor placements in gaussian processes. *Int. Conf. on Machine Learning* (pp. 265–272).

Holub, A., Perona, P., & Burl, M. C. (2008). Entropy-based active learning for object recognition. *Proc. Online Learning for Classification* (pp. 1–8).

Knuth, K., Erner, P., & Frasso, S. (2007). Designing intelligent instruments. *MaxEnt, Amer. Inst. of Physics, Conf. Proc. 954* (pp. 203–211).

Kwakernaak, H., & Sivan, R. (1972). *Linear optimal control systems*. Wiley-Interscience.

Lindenbaum, M., Markovitch, S., & Rusakov, D. (2004). Selective sampling for nearest neighbor classifiers. *Machine Learning*, *54*, 125–152.

MacKay, D. (1992). Information-Based Objective Functions for Active Data Selection. *Neural Computation*, *4*, 590–604.

Olivetti, E. (2008). *Sampling strategies for expensive data*. Doctoral dissertation, U. of Trento.

Pfingsten, T. (2006). Bayesian active learning for sensitivity analysis. *Lec. Notes in C.S.*, *4212*, 353–364.

Platt, J. (1999). Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. *Adv. in Large Margin Classifiers* (pp. 61–74). MIT Press.

Sacks, J., Welch, W., Mitchell, T., & Wynn, H. (1989). Design and Analysis of Computer Experiments. *Statistical Science*, *4*, 409–423.

Tong, S., & Koller, D. (2001). Support vector machine active learning with applications to text classification. *Machine Learning Research*, *2*, 45–66.

Vapnik, V. (1995). *The nature of statistical learning theory*. Springer, New York.

Veeramachaneni, S., Olivetti, E., & Avesani, P. (2006). Active sampling for detecting irrelevant features. *Int. Conf. on Machine Learning* (pp. 961–968).

Zadrozny, B. (2004). Learning and evaluating classifiers under sample selection bias. *Int. Conf. on Machine Learning* (p. 114).

Table 1. Probability of detection, P_d, for the SVM methods on each oracle for $P_{fa} = 0.05$ at round=50.

ID	SVM-PAS	SVM-MCP	SVM-Meta
1	0.918 ± 0.028	0.975 ± 0.005	$\mathbf{0.988} \pm 0.004$
2	0.867 ± 0.037	$\mathbf{0.979} \pm 0.004$	0.951 ± 0.019
3	0.774 ± 0.038	0.886 ± 0.043	$\mathbf{0.908} \pm 0.032$
4	0.713 ± 0.044	0.636 ± 0.033	$\mathbf{0.782} \pm 0.033$
5	0.653 ± 0.054	0.811 ± 0.035	$\mathbf{0.872} \pm 0.029$
6	0.794 ± 0.044	0.843 ± 0.046	$\mathbf{0.851} \pm 0.040$
7	0.612 ± 0.049	$\mathbf{0.775} \pm 0.019$	0.748 ± 0.045
8	0.647 ± 0.047	0.810 ± 0.022	$\mathbf{0.823} \pm 0.021$
9	0.591 ± 0.037	0.561 ± 0.101	$\mathbf{0.602} \pm 0.072$
10	0.603 ± 0.110	$\mathbf{0.866} \pm 0.015$	0.713 ± 0.031
11	0.287 ± 0.046	0.316 ± 0.064	$\mathbf{0.354} \pm 0.059$
12	$\mathbf{0.349} \pm 0.070$	0.268 ± 0.055	0.315 ± 0.054
13	0.238 ± 0.096	0.336 ± 0.144	$\mathbf{0.392} \pm 0.167$

Table 2. Probability of detection, P_d, for the KDE methods on each oracle for $P_{fa} = 0.05$ at round=50.

ID	KDE-PAS	KDE-MCP	KDE-MIP
1	0.880 ± 0.030	$\mathbf{0.988} \pm 0.006$	0.962 ± 0.021
2	0.865 ± 0.045	0.972 ± 0.009	$\mathbf{0.981} \pm 0.005$
3	0.772 ± 0.043	$\mathbf{0.947} \pm 0.018$	0.929 ± 0.019
4	0.713 ± 0.035	0.813 ± 0.048	$\mathbf{0.822} \pm 0.015$
5	0.642 ± 0.047	0.795 ± 0.040	$\mathbf{0.832} \pm 0.033$
6	0.763 ± 0.051	$\mathbf{0.890} \pm 0.015$	0.885 ± 0.017
7	0.588 ± 0.052	$\mathbf{0.731} \pm 0.023$	0.634 ± 0.028
8	0.674 ± 0.055	0.793 ± 0.023	$\mathbf{0.815} \pm 0.015$
9	0.622 ± 0.038	$\mathbf{0.725} \pm 0.035$	0.615 ± 0.035
10	0.603 ± 0.079	0.781 ± 0.050	$\mathbf{0.881} \pm 0.029$
11	$\mathbf{0.264} \pm 0.039$	0.235 ± 0.057	0.204 ± 0.026
12	0.332 ± 0.071	$\mathbf{0.513} \pm 0.036$	0.167 ± 0.028
13	$\mathbf{0.246} \pm 0.079$	0.140 ± 0.054	0.235 ± 0.114

Table 3. Probability of detection, P_d, for the GP methods on each oracle for $P_{fa} = 0.05$ at round=50.

ID	GP-PAS	GP-MCP	GP-MIP
1	0.873 ± 0.027	$\mathbf{0.955} \pm 0.009$	0.929 ± 0.008
2	0.857 ± 0.046	$\mathbf{0.951} \pm 0.012$	0.924 ± 0.012
3	0.768 ± 0.041	$\mathbf{0.890} \pm 0.022$	0.762 ± 0.029
4	0.730 ± 0.033	$\mathbf{0.866} \pm 0.016$	0.709 ± 0.014
5	0.666 ± 0.044	0.810 ± 0.033	$\mathbf{0.841} \pm 0.012$
6	0.744 ± 0.051	$\mathbf{0.878} \pm 0.011$	0.620 ± 0.037
7	0.609 ± 0.050	$\mathbf{0.720} \pm 0.032$	0.531 ± 0.030
8	0.682 ± 0.051	0.820 ± 0.025	$\mathbf{0.834} \pm 0.042$
9	0.631 ± 0.040	$\mathbf{0.758} \pm 0.030$	0.591 ± 0.020
10	0.603 ± 0.079	0.685 ± 0.055	$\mathbf{0.798} \pm 0.026$
11	0.265 ± 0.040	0.250 ± 0.025	$\mathbf{0.385} \pm 0.014$
12	0.334 ± 0.071	$\mathbf{0.474} \pm 0.024$	0.339 ± 0.021
13	$\mathbf{0.243} \pm 0.078$	0.029 ± 0.012	0.217 ± 0.082

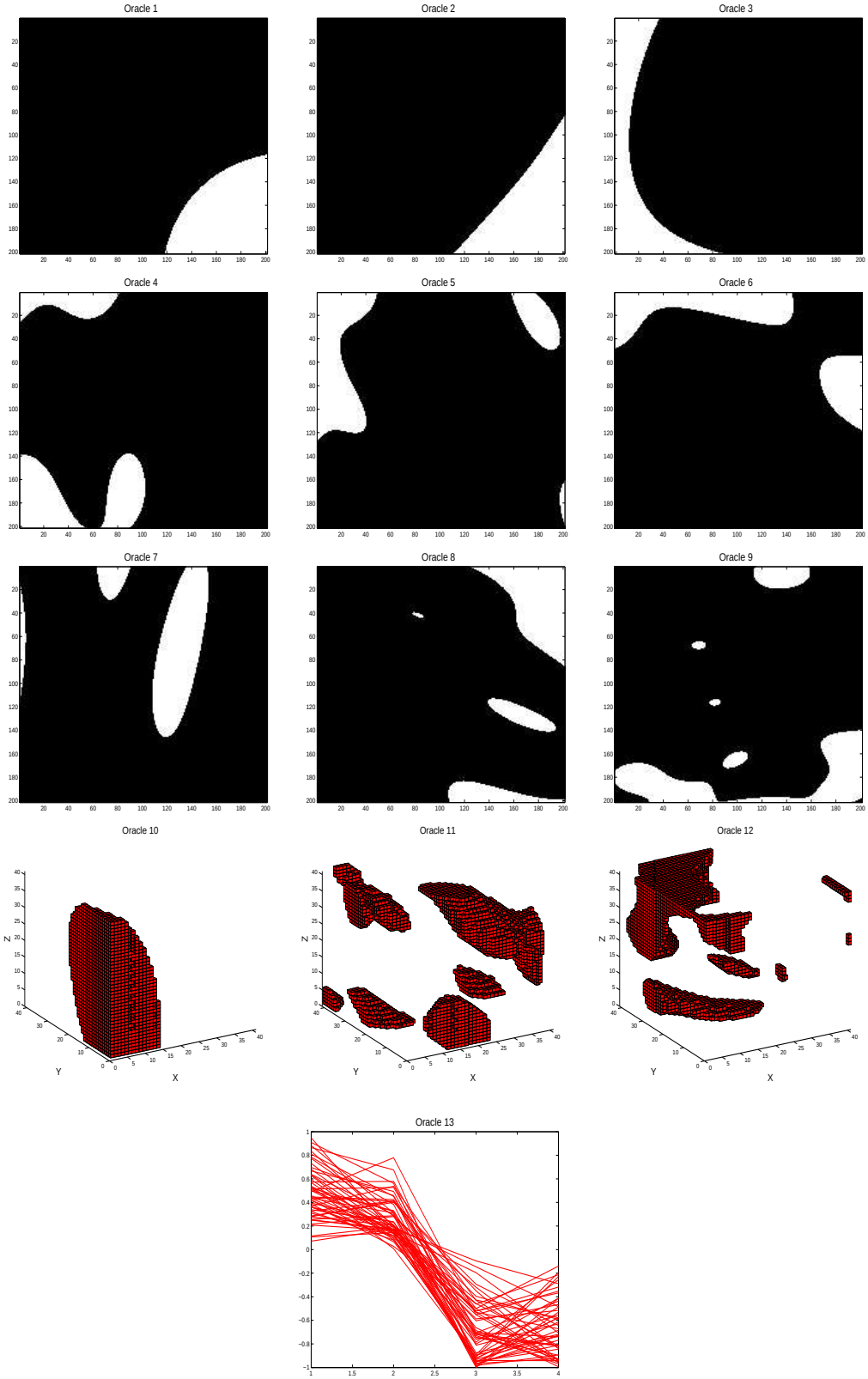

Figure 1. Thirteen oracles were used to evaluate the algorithms. (1)-(9) LevelSet2D ($d = 2$, $P_+ = 0.125$); (10)-(12) LevelSet3D ($d = 3$, $P_+ = 0.062$); (13) Inverted Pendulum ($d = 4$, $P_+ = 0.007$).